

Vom Loggen und Fehler finden Elastic Stack richtig einsetzen

18.05.2020

ObjektForum (OnlineEdition)

Dirk Tröndle

dirk.troendle@andrena.de

Agenda

1. Einführung
2. Stolpersteine bei der Log-Analyse
3. Logzeilen verbessern
 - Loggen im JSON-Format
 - Kontext loggen
4. Best Practices
 - Code
 - Kibana
5. Alternativen

Einführung

▶ Einführung

Früher: Oft monolithische Anwendungen, die auf eigenen Servern bzw. VMs laufen

Heute:

- Vermehrt verteilte Anwendungen, die potentiell auf verschiedenen Servern bzw. in der Cloud laufen
- Kurzlebige Docker-Container, die jederzeit durch andere Container ersetzt werden könnten

▶ Einführung



...

Quelle: <https://www.elastic.co/de/brand>

Stolpersteine bei der Log-Analyse

Stolpersteine bei der Log-Analyse

Elastic Stack ist kein Wunderheilmittel

- Elastic Stack verbessert nicht die Qualität der Logzeilen
- Analyse und Visualisierungsmöglichkeiten sind dadurch beschränkt, wie gut die Logzeilen sind
- Beschränkungen mit klassischen Logzeilen:
 - Visualisierungen sind im Wesentlichen auf Anzahl matchender Strings beschränkt
 - Suchen enthalten oft nur Teile der Message, wenn es in der Anwendung ähnliche Messages gibt, hat man false positives



Stolpersteine bei der Log-Analyse

Logzeilen ohne Kontext, Stacktrace oder Daten die zum Fehler führen

```
2019-05-23 14:14:42.294 ERROR Something went wrong!
2019-05-23 14:14:42.294 WARN  operation failed.
2019-05-23 14:14:42.294 WARN  Transaction failed.
2019-05-23 14:14:42.294 INFO  User operation succeeded.
2019-05-23 14:14:42.294 ERROR Something went wrong!
java.lang.ArrayIndexOutOfBoundsException: 3
    at de.andrena.dtroendle.somePackage.SomeClass.someMethod(SomeClass.java:3)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```



Stolpersteine bei der Log-Analyse

Mischung zwischen Daten und Events in der Logzeile

```
2019-05-23 14:14:42.294 INFO Nutzer mit ID (42) hat sich eingeloggt.  
2019-05-23 14:14:42.294 INFO Buchung [1234] wurde storniert.  
2019-05-23 14:14:42.294 ERROR Rolle mit Id [Admin] existiert nicht.  
2019-05-23 14:14:42.294 WARN Parameter 'xy' is missing. Using default value 'foobar'.
```



Stolpersteine bei der Log-Analyse

```
2019-05-23 14:14:42.294 ERROR User Operation succeeded.
```

```
User Operation succeeded.
```

```
2019-05-23 14:14:42.294 INFO User d.a.d.c.User@27af8038 successfully did something.
```



Logzeilen verbessern

Loggen im JSON-Format

Logzeilen verbessern - Loggen im JSON-Format

Datentypen in den JSON Keys

- Elasticsearch unterstützt die Datentypen die in JSON vorhanden sind
- Die Möglichen Auswertungen von String Values im Log sind stark eingeschränkt
- Mit den anderen Datentypen lassen sich bessere Visualisierungen erstellen, z.B.
 - Durchschnittswerte/Percentile für Werte von Typ Number
 - Histogramme für Werte vom Typ Date
 - Karten für Strings im Locale Format (ISO 3166)
- **Kontext Daten sollten also nicht in die *Message* sondern in eigene Keys!**



Logzeilen verbessern - Loggen im JSON-Format

Einbau in Java - Beispielhafte logback.xml

```
...  
<appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">  
  <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">  
    <providers>  
      <mdc/>  
      <context/>  
      <version/>  
      <logLevel/>  
      <message/>  
      <threadName/>  
      <logstashMarkers/>  
      <arguments/>  
      <stackTrace/>  
      <callerData/>  
    </providers>  
  </encoder>  
</appender>  
...
```

Siehe auch: <https://github.com/logstash/logstash-logback-encoder>



Logzeilen verbessern - Loggen im JSON-Format

Logmessage als Event benutzen

- Substring Suche ist aufwändiger und ist anfällig für false positives
- Wenn die Log-Message nur noch einen konstanten Wert enthält vereinfacht sich die Suche nach konkreten Logzeilen bzw. Events
- Durch das JSON Format kann man trotzdem Daten und Kontext (z.B. Fehlerhafte Daten oder IDs) als weitere Key/Value Paare ergänzen



Logzeilen verbessern - Loggen im JSON-Format

Logmessage als Event benutzen

Vorher:

```
logger.warn("Critical Values detected! SensorId=[{}], AggregationType=[{}], Value=[{}]",  
            sensor.getId(), data.getAggregationType(), data.getValue());
```

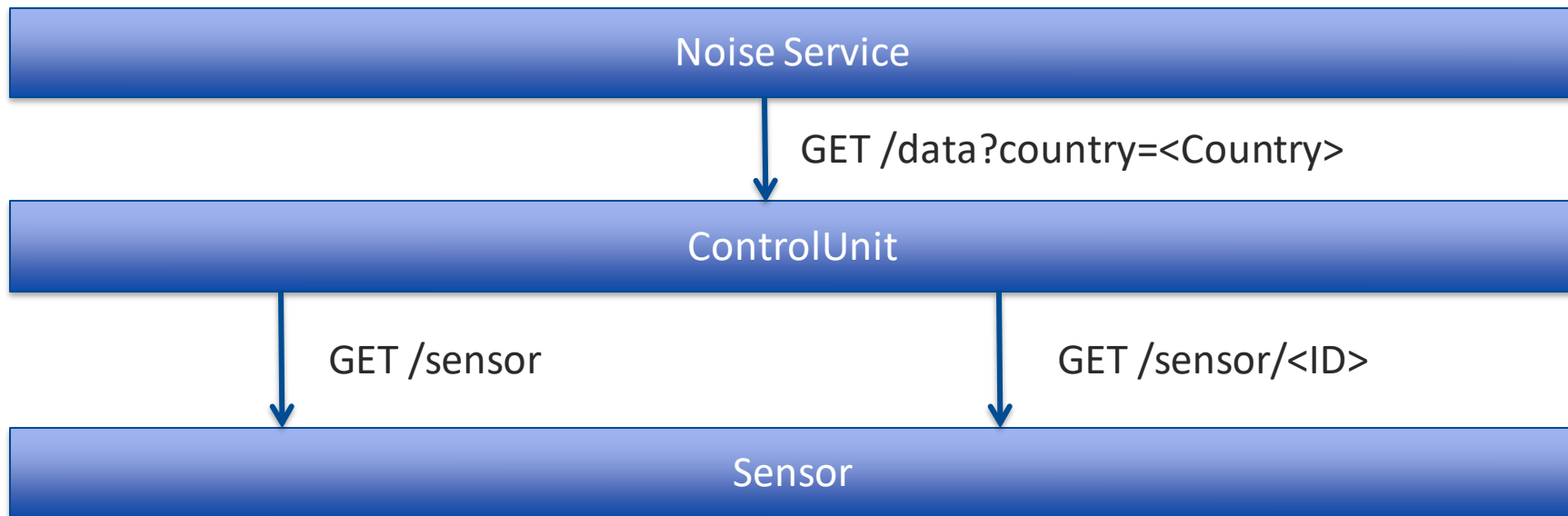
Nachher:

```
import net.logstash.logback.argument.StructuredArguments;  
...  
logger.warn("Critical Values detected!",  
            StructuredArguments.keyValue("sensorId", sensor.getId()),  
            StructuredArguments.keyValue("AggregationType", data.getAggregationType()),  
            StructuredArguments.keyValue("value", data.getValue()));
```



Logzeilen verbessern - Loggen im JSON-Format

Demo: Einsatz in Kibana



Logzeilen verbessern

Kontext loggen

Logzeilen verbessern - Kontext loggen

MDC

- MDC (Mapped Diagnostic Context) ist eine Threadbasierte Map, in der man Werte fürs Logging ablegen kann
- Gibt es in vielen Programmiersprachen
- Ermöglicht zusammengehörige Logzeilen zu erkennen
- Kann den Code der Logzeilen verschlanken



Logzeilen verbessern - Kontext loggen

MDC

- MDC (Mapped Diagnostic Context) ist eine Threadbasierte Map, in der man Werte fürs Logging ablegen kann
- Gibt es in vielen Programmiersprachen
- Ermöglicht zusammengehörige Logzeilen zu erkennen
- Kann den Code der Logzeilen verschlanken
- **Vorsicht:** Man muss hinter sich aufräumen!

```
MDC.clear();
```



Logzeilen verbessern - Kontext loggen

IDs im MDC

- Um zusammengehörige Logzeilen erkennen zu können, kann man eine ID in den MDC legen:

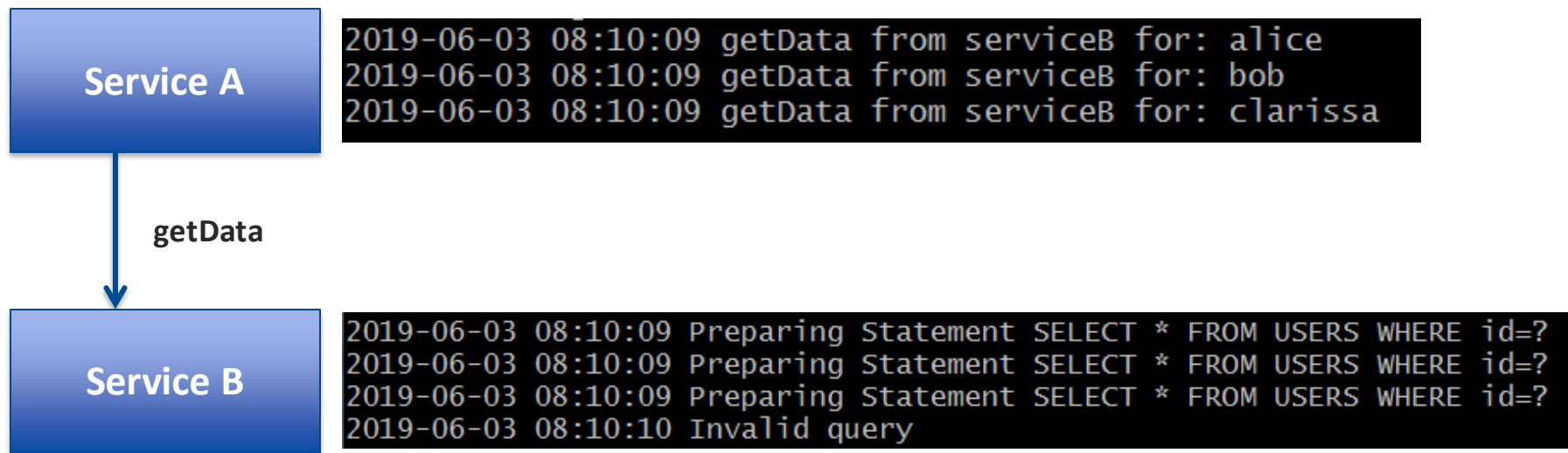
```
MDC.put("loggingId", UUID.randomUUID().toString());
```

- Dies sollte man so früh wie möglich tun, z.B. als RequestFilter oder mit AspectJ
- Es gibt Frameworks, z.B. *Spring Sleuth*, die dies automatisch tun



Logzeilen verbessern - Kontext loggen

Logzeilen über Servicegrenzen hinweg zuordnen



Problem: Welche Logzeile aus Service A gehört zu welcher Logzeile aus Service B?



Logzeilen verbessern - Kontext loggen

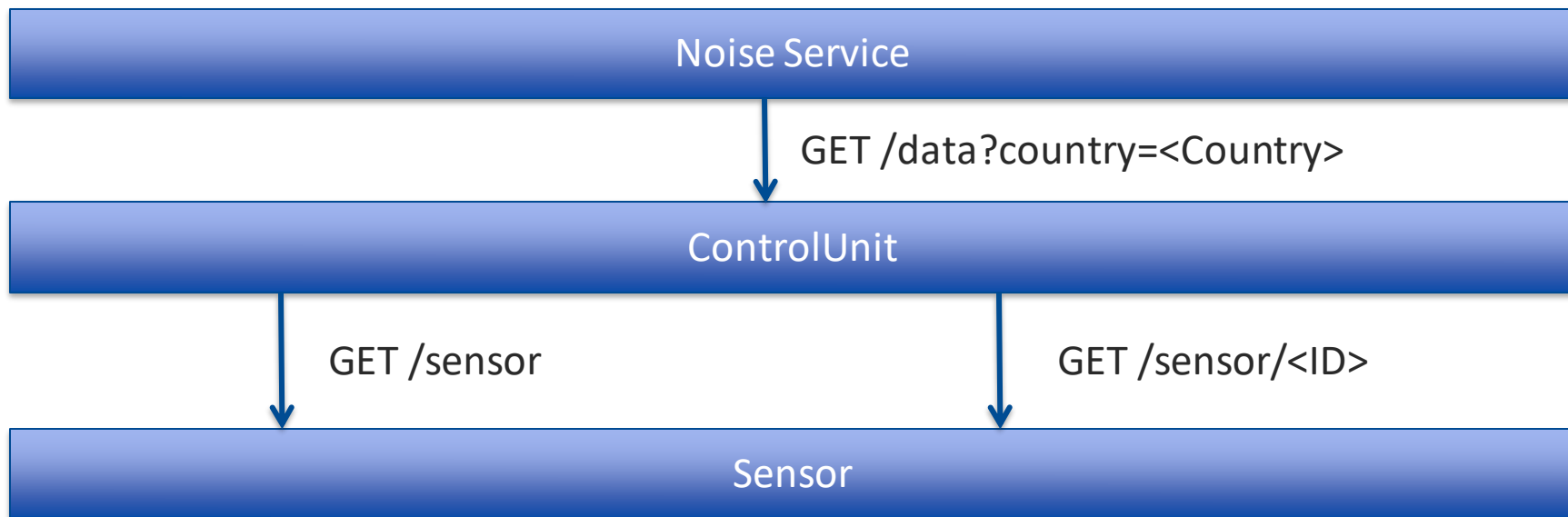
Logzeilen über Servicegrenzen hinweg zuordnen

- Logging-IDs sollten weitergereicht werden, z.B. über Request-Header
- Tools wie *Spring Sleuth* machen dies automatisch
- Wenn Service A in einer Request mehrere Requests an Service B macht, macht es Sinn mehrere IDs weiterzureichen, z.B.
 - Id1: Logging Id im MDC von Service A
 - Id2: Neu generiert bei jeder Request an Service B
 - Service B sollte Id1 und Id2 erhalten und beide im MDC ablegen



Logzeilen verbessern - Kontext loggen

Demo: Fehleranalyse über mehrere Services mit IDs im Log



Logzeilen verbessern - Kontext loggen

Multithreading

Bei Multithreading sollte der MDC an die Kind Threads übergeben werden:

```
Optional<Map<String, String>> callerMdc = Optional.ofNullable(MDC.getCopyOfContextMap());
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        callerMdc.ifPresent(MDC::setContextMap);
        try {
            // child Thread code
        } finally {
            MDC.clear();
        }
    }
};
new Thread(runnable).start();
```


Best Practices

Best Practices

Im Code

- Message kurz, sprechend und möglichst konstant. Am besten Eventname
- Daten in möglichst passendem Datenformat in eigenen Key loggen
- Gleichnamige Keys sollten immer den gleichen Datentyp als Wert haben
- Gleichwertige Keys sollten den gleichen Namen haben
- Falls man immer wieder die gleichen Daten loggt, sollten diese (kurzfristig) ins MDC
- Positiv Logging kann bei hoher Last die Performanz beeinflussen
- Sensible Werte sollten pseudonymisiert bzw. anonymisiert werden



Best Practices

Im Kibana

- Dashboards nicht überladen, nur wichtige Infos/Visualisierungen ins Dashboard
- Falls man Alerts nutzt, sollte die Mail möglichst Präzise auf eine Kibana URL (Visualisierung/Discover) verlinken, auf der man die Ursache direkt sieht
- Zu viele Filter führen zu einem URL-Overflow und machen Dashboards langsamer
- Falls man die Daten sinnvoll trennen kann (z.B. verschiedene Länder), bietet es sich an im Dashboard einen Schalter (Controls-Visualisierung) einzubauen
- Alle Suchen und Visualisierungen sollten getestet werden
 - Auf Testumgebung Logzeilen verursachen
 - Ggf. Spezialversion des Service auf eine Testumgebung deployen, in der die Logzeilen einfach triggerbar sind



Alternativen

Logstash Filter

Alternativen – Logstash Filter

- **Vorteil:**
 - Man kann Logzeilen von Services, die man nicht ändern kann/möchte auch sinnvoll strukturieren
 - Man kann mehrere verschiedene Filter hintereinander schalten
 - Man kann den Typ von Feldern ändern
- **Nachteil:**
 - Die Filter greifen für alle Logzeilen sofern man nicht durch Bedingungen einschränkt
 - Je nach Infrastruktur darf man die Logstash Konfiguration nicht anpassen



Alternativen – Logstash Filter

Kv filter plugin

- Greift bei Logzeilen, die "key=value" enthalten
- Erstellt neues Key-Value Paar im JSON Event
- Klammern oder definierbare Zeichen können automatisch entfernt werden

Doku: <https://www.elastic.co/guide/en/logstash/current/plugins-filters-kv.html>



Alternativen – Logstash Filter

Dissect filter plugin

- Greift bei Logzeilen die dem angegebenen Pattern entsprechen, z.B.:

```
dissect {  
  mapping => {  
    "message" => "%{+ts} %{+ts} %{level} 1 --- [%{thread}] %{logger} : %{msg}"  
  }  
}
```

- Splittet die das angegebene Feld des LogEvents (z.B. "message") in beliebige neue Key-Value Paare
- Lohnt sich wenn sich ähnlich formatierte Logzeilen oft wiederholen

Doku: <https://www.elastic.co/guide/en/logstash/current/plugins-filters-dissect.html>



Alternativen – Logstash Filter

Grok filter plugin

- Ermöglicht Erzeugung von typisierten Key-Value Paaren via regex match
- Viele Patterns sind vordefiniert, z.B. TOMCATLOG
- Eigene Patterns sind auch definierbar
- Kann sinnvoll in Kombination mit dissect eingesetzt werden,
 - gleichbleibende Teile mit dissect aufteilen
 - wechselnde Teile nachgelagert mit grok

Doku: <https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html>



Alternativen

Ohne StructuredArguments von logback

Alternativen – ohne StructuredArguments von logback

Java Anwendungen

- Ab slf4j 2.0.0 (aktuell in alpha):
 - Fluent API mit keyValue Möglichkeit

```
logger.atDebug()  
    .addKeyValue("oldT", oldT)  
    .addKeyValue("newT", newT)  
    .log("Temperature changed.");
```



Alternativen – ohne StructuredArguments von logback

Mit MDC Support – LoggerWrapper der MDC benutzt

```
public class LoggerWrapper {  
  
    private final Logger logger;  
  
    private LoggerWrapper(Logger logger) {  
        this.logger = logger;  
    }  
  
    public void info(String event, Map<String, String> structuredValues) {  
        // add structuredValues to MDC  
        structuredValues.forEach((key, value) -> MDC.put(key, value));  
        // log the constant string as message  
        logger.info(event);  
        // remove structuredValues from MDC  
        structuredValues.forEach((key, value) -> MDC.remove(key));  
    }  
  
    public static LoggerWrapper getLogger(Class<?> clazz) {  
        return new LoggerWrapper(LoggerFactory.getLogger(clazz));  
    }  
  
}
```



Vielen Dank für die Aufmerksamkeit!

Gibt es Fragen?